SciSpot: A Framework for Low-cost Scientific Computing On Transient Cloud Servers

Paper # 105

ABSTRACT

In this paper, we...

1 INTRODUCTION

Scientific computing applications are a crucial component in the advancement of science and engineering, and play an important role in the analysis and processing of data, and understanding and modeling natural processes. These applications are typically implemented as large-scale parallel programs that use parallel-computing communication and coordination frameworks such as MPI. To take advantage of their parallel nature, conventionally, these applications have mostly been deployed on large, dedicated high performance computing infrastructure such as super computers.

(Vikram: Application of computer simulation plays a critical role in understanding natural and synthetic phenomena associated with a wide range of material, biological, and engineering systems. This scientific computing approach involves the analysis and processing of data generated by the mathematical model representations of these systems implemented on computers. Typically, these scientific computing applications (simulations) are designed as parallel programs that leverage the communication and coordination frameworks associated with parallel computing techniques such as MPI in order to yield useful information at a faster pace (shorter user time). To exploit the parallel processing capabilities, such applications are routinely deployed on large, dedicated high performance computing infrastructure such as supercomputers [? ? ?].)

Increasingly, cloud computing platforms have begun to supplement *cite* and complement *cite; how true is this?* conventional HPC infrastructure ~~in order~~ to meet the large computing and storage requirements of scientific applications. Public cloud platforms such as Amazon's EC2, Google Cloud Platform, and Microsoft Azure, offer multiple benefits such as *on-demand* resource allocation, convenient pay-as-you-go pricing models, ease of provisioning and deployment, and near-instantaneous elastic scaling. Most cloud platforms offer *Infrastructure as a Service*, and provide computing resources in the form of *Virtual Machines (VMs)*, on which a wide range of applications such as web-services, distributed data processing, distributed machine learning, etc., are deployed.

(Vikram: The extensive use of cloud platforms to host and run a wide range of applications such as web-services and distributed data processing have inspired early investigations in the direction of using these resources for scientific computing applications to meet the large computing and storage requirements of the latter. Early work in this area has shown the potential of using these cloud platforms to both supplement and complement conventional HPC infrastructure. Public cloud platforms such as Amazon's EC2, Google Cloud Platform, and Microsoft Azure, offer multiple benefits: *on-demand* resource allocation, convenient pay-as-you-go pricing models, ease of provisioning and deployment, and near-instantaneous elastic scaling, to name a few. Most cloud platforms offer *Infrastructure as a Service*, and provide computing resources in the form of Virtual Machines (VMs) that are application-agnostic and can serve as deployment sites for a wide range of applications such as web-services, distributed machine learning, and scientific simulations.)

, ,
.

~~In order to~~ To meet the diverse resource demands of different applications, public clouds offer resources (i.e., VM's) with multiple different resource configurations (~~such as~~e.g., number of CPU cores~~, and~~ memory capacity), and pricing and availability contracts. Conventionally, cloud VMs have been offered with "on-demand" availability, such that the lifetime of the VM ~~was~~is solely determined by the owner of the VM (i.e., the cloud customer). Increasingly however, cloud providers have begun offering VMs with *transient*, rather than continuous on-demand availability. Transient VMs can be unilaterally revoked and preempted by the cloud provider, and applications running inside them face fail-stop failures. Due to their volatile nature, transient VMs are offered at steeply discounted rates. Amazon EC2 spot instances, Google Cloud Preemptible VMs, and Azure Batch VMs, are all examples of transient VMs, and are offered at discounts ranging from 50 to 90%.

However, deploying applications on cloud platforms presents multiple challenges due to the *fundamental* differences with conventional HPC clusters—which most applications still assume as their default execution environment. While the on-demand resource provisioning and pay-as-you-go pricing makes it easy to spin-up computing clusters in the cloud, ~~for effective resource utilization,~~ the deployment of applications on cloud platforms must be cognizant of the heterogeneity in VM sizes, pricing, and availability for effective resource utilization. Crucially, optimizing for *cost* in addition to~~, and not just~~ makespan~~,~~ becomes an important objective in cloud deployments. Furthermore, although using transient resources can drastically reduce computing costs, their preemptible nature results in frequent job failures. Preemptions can be mitigated with additional fault-tolerance mechanisms and policies [27**?**], although they impose additional performance and deployment overheads *such as? needs one more sentence to clearly explain why these existing approaches are not addressing the need that we address in this paper below....*

(Vikram: While the on-demand resource provisioning and pay-as-you-go pricing makes it easy to spin-up computing clusters in the cloud, the deployment of applications on cloud platforms must be cognizant of the heterogeneity in VM sizes, pricing, and availability for effective resource utilization. Crucially, optimizing for *cost* in addition to makespan, becomes an important objective in cloud deployments. Furthermore, although using transient resources can drastically reduce computing costs, their preemptible nature results in frequent job failures. Preemptions can be mitigated with additional fault-tolerance mechanisms and policies [27**?**], although they impose additional performance and deployment overheads. *add one more sentence.* These considerations of cost, server configuration heterogeneity, and frequent job failures intrinsic to the system present multiple challenges in deploying applications on cloud platforms which are fundamentally different from those that appear in using HPC clusters as the execution environment for the scientific computing applications.)

In this paper, we develop principled approaches for deploying and orchestrating ~~parallel~~ scientific computing applications on the cloud ~~at low cost~~, and present SciSpot, a ~~system~~framework for low-cost scientific computing on ~~cloud~~ transient cloud servers. Our policies for tackling the resource heterogeneity and transient availability of cloud VMs build on a key insight: most scientific computing applications are deployed as a collection or "bag" of jobs. These bags of jobs represent multiple instantiations of the same computation with different parameters. For instance, each job may be running a (parallel) simulation with a set of ~~simulation~~input parameters, and different jobs in the collection run the (same) simulation ~~on~~employing a different set of input parameters. Collectively, a bag of jobs can be used to "sweep" or search across a multi-dimensional parameter space to discover or narrow down the set of ~~feasible and viable~~ viable and/or interesting parameters associated with the modeled natural or synthetic processes. A similar approach is adopted in the use of machine learning (ML) to enhance scientific computational methods, a rapidly emerging area of research, when a collection of jobs with independent parameter sets are launched to train ML models to predict simulation results and/or accelerate the simulation technique.

Prior approaches and systems for mitigating transiency and cloud heterogeneity have largely targeted individual instantiations of jobs [27, 29, 32**?**]. For a bag of jobs, it is not necessary, or sufficient, to execute an individual job in timely manner—instead, we could selectively restart failed jobs in order to complete the necessary, desired subset ~~a~~ ~~*fraction*~~ of jobs in a bag. Furthermore, treating the bag of jobs as a fundamental unit of computation allows us to select the "best" server configuration for a given application, by exploring different servers for initial jobs and running the remainder of the jobs on the optimal server configuration *is this optimal server found on the fly after the initial set of explorations, or this can be found separately?*.

We show that optimizing across an entire bag of jobs and being cognizant of the relation between different jobs in a bag, can enable simple and powerful policies for optimizing cost, makespan, and ease of deployment. We implement these policies as part of the SciSpot framework, and make the following contributions: *skipping the following contributions, will look at it after they are reordered and further refined*

(1) In order to select the "right" VM from the plethora of choices offered by cloud providers, we develop a search-based server selection policy that minimizes the cost of running applications. Our search based policy selects a

transient server type based on it's cost, parallel speedup, and probability of preemption.
(2) Since transient server preemptions can disrupt the execution of jobs, we present the *first* empirical model and analysis of transient server availability that is *not* rooted in classical and out-dated bidding models for EC2 spot instances that have been proposed thus far. Our empirical model allows us to predict expected running and costs of jobs of different types and durations.
(3) We develop preemption-mitigation policies to minimize the overall makespan of bags of jobs, by taking into consideration the partial redundancy and relative "importance" of different jobs within a bag. Combined, our policies yield a cost saving of XXX% and a makespan reduction of XXX% compared to conventional cloud deployments, and a makespan reduction of XXX% compared to a conventional HPC supercomputer.
(4) Finally, ease of use and extensibility are one of the "first principles" in the design of SciSpot, and we present the design and implementation of the system components and present case studies of how scientific applications such as molecular dynamics simulations can be easily deployed on transient cloud VMs.

2 BACKGROUND

2.1 Transient Computing

2.2 Case Studies: Bags of Jobs in Scientific Computing Applications

For testing and evaluating the SciSpot framework, we consider three representative examples (case studies) from molecular dynamics (MD) and hydrodynamics simulations: 1) MD simulations of ions in nanoscale confinement created by material surfaces [19, 22], MD-based optimization dynamics of shape-changing deformable nanoparticles (NPs) [17, 18], and hydrodynamics simulations of continuum material models using the Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) code [23, 24]. These examples are representative of typical scientific computing applications in the broad domain of physics, materials science, and chemical engineering; the first two applications (1 and 2) are based on codes and associated theoretical formulations developed by us [15–19, 30], and case study 3 is based on an open-source code developed at Lawrence Livermore National Laboratories [1, 23].

The typical workflow associated with most scientific computing applications, including the aforementioned case studies, involves the implementation of the "bags of jobs" approach at many critical stages. In the initial stage, the construction and calibration of the appropriate model often involves testing for the needed attributes (e.g., characteristic sizes, interactions potentials) of the building blocks (model components) by sweeping over different combinations of physical as well as computing parameters (e.g., simulation timestep, thermostat variables) and eliminating the sets that lead to unphysical, unstable, or computationally intractable scenarios. During the model examination stage for the investigation of the accuracy and generalization of the model to describe the associated natural or synthetic processes, the dynamics of the model system is simulated over a wide range of model parameters. Accordingly, multiple sets of simulations (bags of jobs) are run to sweep over a broad region of the multidimensional parameter space and to isolate the domains where the model works best and where it yields a poorer representation of the real system.

Often, the key objective of the scientific computing application is to isolate the model system parameters where interesting changes in the material structure or assembly behaviors (e.g., phase transitions) are observed. A similar bags of jobs approach is also adopted in such applications with the search for these model parameters generally inspired by experimentally-informed observations and/or predictions yielding from approximate analytical theoretical formulations. For example, in the simulation of deformable nanoparticles implemented in the NP shape code, one is interested in isolating the set of NP and environmental parameters: NP bending modulus, NP stretching constant, NP charge, and salt concentration, that yields complex NP deformations/shapes (e.g., discs, rods, bowls). Similarly, in the ions in nanoconfinement application, a quantity of interest is the set of electrolyte system attributes (parameters): confinement length h, positive ion valency (z_p), negative ion valency (z_n), electrolyte concentration c, and ion diameter d, that yields the expected contact density or the experimentally-measured effective pressure between the confining nanomaterial surfaces. Finally, the bags of jobs approach is adopted during the completion process in the workflow where simulations are often launched in parallel to fill any gaps in the extracted trends or to obtain error bars on the predictions (e.g., ionic density profiles, energy distributions, NP shape transitions).

In addition to the conventional scientific computing (HPC) applications, an emerging area of research in a broad range of fields including materials science, biology, neuroscience, and physics where the bags of jobs approach is critical to the workflow is the integration of machine learning (ML) tools with these HPC applications [3, 7, 9, 10, 20, 21, 25, 26, 28, 35]. ML methods have been developed and implemented to identify model attributes/parameters that yield desirable material configurations [31], update configurations in simulations [4, 25], predict and auto-tune optimal simulation control parameters [20], predict critical features associated with simulation output [21], infer assembly landscapes [9, 26], and classify phases of matter [7]. In many of these examples,

ML models (e.g., artificial neural network, support vector machines) are trained on large data sets generated via simulations run over a broad range of parameter values. The bags of jobs process is invoked multiple times during the experimentation with many ML techniques using training and testing datasets to isolate the ML method(s) that yield the most accurate results for a given scientific computing application. For example, in Ref. [21], an ANN was trained to predict contact, peak, and mid-point ionic density using training and testing datasets comprising of $\approx$ 4800 and $\approx$ 2000 simulation runs respectively. Datasets were generated using HPC resources (Bigred2 computing cluster) based on the sweep of parameters $h \in (3.0, 4.0)$ nm, $z_p \in 1, 2, 3$ (in units of electronic charge $|e|$); $z_n \in -1, -2$ $|e|$; $c \in (0.3, 0.9)$ M, and $d \in (0.5, 0.75)$ nm. We envision the SciSpot framework described here to complement and supplement conventional HPC supercomputer systems in enabling the construction of such ML layers (wrappers) [10, 21] around scientific computing applications.

3 PREEMPTION DYNAMICS OF TRANSIENT CLOUD SERVERS

In order to understand and improve the performance of applications running on transient cloud servers, we must understand the nature and dynamics of their preemptions. The preemption characteristics are governed by the supply of surplus resources, the demand for cloud resources, and the resource allocation policies enforced by the cloud operator. Therefore, in this section, we present empirical and analytical models to help us understand the nature of preemptions.

3.1 Price based preemption models

Amazon's EC2 spot instances were the original cloud transient servers. The preemptions of EC2 spot instances is based on their *price*, which is dynamically adjusted based on the supply and demand of cloud resources. Spot prices are determined based on a continuous second-price auction, and if the spot price increases above a pre-specified maximum-price, then the server is preempted.

Thus, the time-series of these spot prices can be used for understanding preemption characteristics such as the frequency of preemptions and the "Mean Time To Failure" of the spot instances. Many research projects have used publicly available[1]historical spot prices to characterize and model spot instance preemptions [**?** **?**]. For example, past work has analyzed spot prices and shown that the MTTF's of spot instances of different hardware configurations and geographical zones ranges from a few hours to a few days [**?** **?**].

[1]Amazon posts Spot prices of 3 months, and researchers have been collecting these prices since 2010 [**?**].

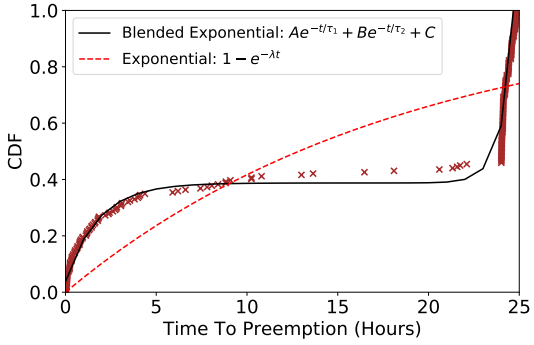

Figure 1: CDF of lifetimes of Google Preemptible Instances. Our blended exponential distribution fits much better than the conventional exponential failure distributions.

However, Amazon has recently changed the preemption characteristics of spot instances, and servers are now preempted even if the spot price is below the maximum price. Thus, spot prices are no longer a completely reliable indicator of preemptions, and preemptions can no longer be inferred from looking at prices alone. Therefore, new techniques are required to model preemption dynamics that can supplement the earlier price-based approaches, and we develop these techniques next.

3.2 Empirical preemption models

The preemptions of transient servers need not be related to their price. For example, Google's Preemptible VMs and Azure Batch VMs have a *fixed* price relative to their non-preemptible counterparts. In such cases, price based models are inadequate, and other approaches to understand preemptions is required.

This task is further complicated by the fact that these cloud operators (Google and Microsoft) do not currently provide any information about preemption characteristics. Thus, relatively little is known about the preemptions (and hence the performance) of these transient VMs.

In order to understand preemption dynamics of transient servers, we conduct a large-scale empirical measurement study which is the first of its kind. We launched more than 1000 Google Preemptible VMs of different types over a two month period (Feb–April 2019), and measured their time to preemption (aka, their useful lifetime).[2]

A sample of 100 such preemption events are shown in Figure 1, which shows cumulative distribution of the VM lifetimes. Note that the cloud operator (Google) caps the *maximum* lifetime of the VM to 24 hours, and all the VMs

[2]We will release the complete preemption dataset and hope that other researchers can benefit.

are preempted before that hard limit. Furthermore, the lifetimes of VM's are *not* uniformly distributed, but have three distinct phases. We observe that many VM's are quickly preempted after they are launched, and thus have a steep rate of failure initially. The failure rate is "bath tub" shaped, with VM's that survive past 3 hours enjoying a relatively low preemption rate, and finally a steep increase in the number of preemptions as the preemption deadline (24 hours) approaches.

We note that this preemption behavior, imposed by the small, 24 hour lifetime, is *substantially* different from conventional failure characteristics of hardware components and even EC2 spot instances. In these "classical" setups, the rate of failure is usually modeled using an exponential distribution, such that $f(t) = \lambda e^{-\lambda t}$, where $\lambda = 1/\text{MTTF}$. However, Figure 1 also shows the CDF ($= 1 - e^{-\lambda t}$) of the exponential distribution when fitted to the observed preemption data, by finding the distribution parameter λ that minimizes the least squares error. From Figure 1, we can see that the classic exponential distribution is unable to model the observed preemption characteristics. The primary reason is that the exponential distribution assumes that the preemptions are *memoryless*, which does not hold true when there is a fixed upper bound on the lifetime, as is the case for Google Preemptible VMs.

3.3 Analytical modeling of preemptions in Google cloud

In order to better understand and characterize preemptions, we now develop a general analytical model for preemption dynamics that is faithful to the empirically observed data and that provides a basis for developing running-time and cost minimizing optimizations that we present in Section **??**.

Our model is based on our earlier observation that the distribution of cumulative lifetimes has multiple distinct temporal phases. In order to model the overall empirical CDF, we develop a *new* probability distribution that is composed by blending two failure processes that act on different temporal phases over the 24 hour maximum lifetime of the VMs.

We write the general form of our blended preemption CDF as follows:

$$F_{\text{Blended}}(t) = A(1 - e^{-t/\tau_1} + e^{\frac{t-b}{\tau_2}}) \quad (1)$$

For most of it's life, a VM sees failures according to the convential exponential distribution with a rate of failure equal to $1/\tau_1$, which is the first term in 1. However, this does not capture the finite lifetime of the VM imposed by the cloud operator. As VMs get closer to their maximum lifetime (24 hours), they are reclaimed (i.e., preempted) at a high, exponential rate, which is captured in the second term of the CDF ($e^{\frac{t-b}{\tau_2}}$). However, this exponential reclamation is only applicable towards the end of the VM's maximum lifetime, and thus the $(t - b)$ term helps to ensure that it does not dominate over the entire temporal range. As before, $1/\tau_2$ is the rate of the reclamation.

This "blended exponential" distribution thus captures the different phases of the preemption CDF through parameters τ_1, τ_2, b, and the parameter A is used to ensure we meet the boundary conditions ($F(0) = 0, F(24) = 1$) These parameters can be obtained for a given empirical CDF by minimizing least-squared function fitting methods. [3]

In the next section, we use the blended exponential analytical model for optimizing cloud resource selection such that we can run scientific applications at low cost and running times.

Optional text. Spoils flow currently Preemption dynamics can also be analyzed through the *survival rates* of VMs. Let $S(t)$ denote the number of VMs that survive till time t. Then, without loss of generality: $F(t) = A(1 - S(t))$

In the case of the blended exponential distribution, $S(t)$ depends on $S(t)$ itself, as well as the failure rates of different phases and the current lifetime (t), as follows:

$$\frac{dS(t)}{dt} = \{\frac{1}{\tau_2} - \frac{1}{\bar{\tau}}(\frac{1}{1 - e^{-b/\tau_2} \cdot e^{t/\bar{\tau}}})\}S(t) \quad (2)$$

4 SCISPOT DESIGN

Scientific simulation applications consume a large amount of computational resources, and are often used in the context of *exploratory* research, where a large amount of jobs are run with different simulation parameters. This can be either a *parameter sweep*, where a large number of parameters need to be evaluated, or a *search* over a large parameter space for the "right" set of parameters that yield the desired model behavior.

In this paper, we look at the problem of running scientific simulations on *transient* computing resources in public clouds.

Past work has largely been focused on running parallel jobs (such as MPI) in the cloud. However, considering entire *job-groups* or ensembles of jobs presents new challenges and opportunities in timely, low-cost computation.

Our system, SciSpot, is a unified framework for running large job-groups that result from parameter exploration.

Input and some assumptions: We assume that the job-group consists of $J_1 \ldots J_N$, with each job evaluating a model on some parameter. The list of parameters to explore can either be generated apriori (as in the case of parameter sweeps), or be dynamically generated as in the case of a search.

In this section, we will look at how we address these challenges:

[3] More details about the distribution fitting are presented in the implementation section(**??**

(1) How to select the right type of cloud server for an application?
(2) How to effectively run job-groups?

SciSpot's key insight is considering an entire job-group can allow better and simpler optimizations that can be easily deployed.

Job-groups are executed in two phases. In the first phase, we search for the right type of server for the jobs in the job-group, and then in the second phase, we execute the remaining jobs on the chosen servers.

4.1 High-level flow

SciSpot is as a cost aware cloud resource manager for running large collections of parallel jobs. The collection forms a "bag" of jobs, which each job running the same executable but with different input parameters. As explained earlier, this is a common use case.

Running a large collection of computationally similar jobs permits many cost and performance optimizations. Given that cloud platforms offer a large number of resource configurations for their VMs, selecting the "right" VM for jobs can be especially beneficial in reducing the cost and running times. Running a large number of jobs with similar computation, communication, and runtime characteristics allows us to "explore" the right server configuration.

The execution of a bag of job kicks off with the user providing the executable, the expected resources requirements for a single job, and the fraction of jobs that must be completed.

Generating Jobs For a Bag: Optionally, SciSpot can also take as input a description of the (multi-dimensional) parameter space, and then generate the jobs. We do this with the users specifying the values that different job parameters can take, and then producing a list with all the permutations. Because job-failures can cause some parameter combinations to remain unexplored, we strive for uniformity of sampling by randomizing the order in which jobs in a bag are scheduled, so that we don't end up in a situation with a large fraction of any parameter remaining unexplored if the completion threshold has been met.

More details are provided in the Interface and Implementation section.

Therefore, SciSpot's execution of a bag of jobs is composed of two serial phases. In the first phase, we explore different cluster configurations to find the lowest cost server type. In the second phase, we run the reminder of the jobs in the bag on the servers of the selected type (the "exploitation" phase).

4.2 Trade-offs in Server Selection

This presents us with many challenges in the cloud-deployment of these jobs.

Cloud providers offer multiple types of instances (VMs), with different hardware configuration (such as number of CPUs and memory size). The price of cloud servers is related to their hardware configuration, but it may not be strictly proportional to the hardware performance. For example, a VM with 32 CPUs may not be 32 times the cost of a single CPU VM.

For parallel and distributed applications, the type of servers selected has large implications on their performance. Consider the case of deploying an application on 8 8-core VMs vs. 16 4-core VMs. In both cases, the total number of CPU cores is the same. However, the larger number of VMs requires more communication between the application tasks, and thus may result in performance degradation. The performance of applications at different cluster configurations depends on their communication patterns and scaling properties.

Thus, when deploying applications on the cloud, one has to mindful of the cost and performance tradeoff. However, in the case of transient servers, the story does not stop here.

In addition to pricing differences, the transient availability of instances *also* differs by type. Because the availability of a transient VM is broadly determined by the overall supply and demand of the instances of that *particular* type, the "preemption rate" of VMs often depends on the type of the instance.

Thus, selecting a transient cloud server involves a complex tradeoff between the cost of servers, their performance, and the preemption-rate. We develop server selection policies in the next section.

Figure with different CDF's here.

4.3 Exploration-based Server Selection

SciSpot's server selection policy seeks to identify the best server type for a given job-group.

As stated in the previous subsection, the transient server selection problem is challenging because it involves balancing multiple optimization criteria: applications want low cost, low preemptions, and high performance.

Server selection based on application characteristics is a subject of a growing amount of recent work. These approaches often use micro benchmarks to gather performance data of cloud servers, and then use application performance models to determine suitable VMs for a specific application. Another class of approaches uses "black box" performance modeling, where the application's performance is modeled using a function of the resources, for example, by using linear regression.

In contrast to prior work, our server selection employs a "cold start" policy, and we do not run profiling or pilot jobs that can increase the overall running time and cost. Instead, we search for the "best" cluster configuration for jobs in

a job-group, by exploring the cluster configuration space for the "optimum" server type that optimizes all the desired parameters: cost, running time, and revocation rate.

Thus, the first e jobs in the job group $J_1 \ldots J_e$ are the exploration jobs, run on different cluster configurations. We limit the total number of combinations to explore, by allowing users to submit an estimate of the total number of CPU cores that they desire for each job. This allows us to meet the user expectations in terms of performance and cost—whether the user expects us to spend a large amount of resources or not.

Thus, assuming that there are s different types of VM instances, the first s jobs are run on the s different types. Note that we use homogeneous clusters, since the performance of BSP programs in Heterogeneous environments can be degraded, and importantly, as we show, there are no performance or cost benefits to Heterogenity.

For each server type i, we calcuate the expected cost $E[C_i]$.

$E[C_i] = n_i * c_i * E[T_i]$, where c_i is the price (per second) of the server, n_i is the number of servers of that type required to meet the core-count requirement. The expected running time of the job depends on two factors: the actual running time T, and the increase in running time due to preemptions. Each preemption is akin to a fail-stop failure, which requires an application to restart. Our system makes no assumptions about the fault-tolerance policies supported by the application. For example, some applications may be able to *checkpoint* their state periodically. In either case (checkpointing or not), there is some work lost due to revocations. For ease of exposition, we assume no checkpointing. We discuss checkpointing in the next section (or never?)

Expected running time: Let the running time without failure be T:

$$E[T_i] = T + P(\text{at least one failure}) * T/2 \tag{3}$$

For calcuating the probability of failure, we assume that the failure rate of an individual server of the type is p_i.

$$P(\text{at least one failure}) = 1 - P(\text{no failure}) \tag{4}$$

$$= 1 - (1 - p_i)^{n_i} \tag{5}$$

Thus, we can see that if we select smaller VMs, we will require more of them (higher n_i), and this cluster configuration will have a larger probability of failure and thus higher running times and costs.

The probability of failure p_i depends on the type of server, and we use historically determined failure distributions. Roughly, if we assume exponentially distributed failures, then:

$$p_i = \frac{T}{\text{MTTF}_i} \tag{6}$$

../graphs/cdf_comparison_3.pdf

Where MTTF is the mean time to failure of the server type, and T is the empirically determined job running time without failures (the best case).

In addition to searching over the servers, the effective number of servers is also dynamic in the case of transient environments due to preemptions. Thus, once we have found the appropriate server, we then explore the application's performance at smaller cluster sizes, which helps in the job-group policies that we discuss next.

4.4 Preemption-handling Policies

We run the remaining jobs on the right configuration that is determined through the server selection policy.

Our goal is to minimize costs given a deadline.

This determines two things: how many jobs should be run in parallel, and what to do upon a revocation.

The number of jobs in parallel determines the overall size of our cluster.

$$\text{number of parallel jobs} = \frac{N \cdot T}{\text{Deadline Duration}}$$

If a server is preempted, then the job running on it will cease to run. Our preemption handling policies then decide what to do:

(1) Restart the job on a smaller number of servers
(2) Replenish lost servers and restart job
(3) Discard job. This may be useful in case of parameter sweeps.

In addition, the user is also allowed to provide the fraction of jobs that are allowed to fail (η).

We exploit the naturally occuring intra-job redundancy.

New jobs. For new jobs, the questions are similar to the preemption policy ones, because they also must take into account the failures.

That is, assume that a job has finished and it is time to run a new job. So now, we have a set of servers that successfully ran a job. If we run on same set, then we may hit the 24 hour wall. But, if we discard these and launch new ones, then we face the infant mortality problem. So the question is, at what age should servers be retired? If they are too close to EOL (24 hours), then what's the point? Better start something fresh.

4.5 Checkpointing

Checkpointing requires the same number of servers, which may be tricky, whereas restarts can be on smaller number of nodes no problem.

Maybe talk about the dynamic programming based checkpointing here?

4.6 Early Stopping

Based on the energy function, we can stop some simulations early. The early stopping criteria helps in minimizing the number of jobs run to completion.

We use this to proactively monitor jobs, as well as to decide whether to restart a job if it is preempted.

This can be a fairly substantial section

5 CHECKPOINTING POLICIES

High rate of failure means that especially for long jobs, checkpointing is necessary. In this section, we develop checkpointing policies.

At a high level, we only checkpoint long jobs, based on the probability of preemption.

Once checkpointing for a job is enabled, we use proactive periodic checkpointing.

DMTCP is used for checkpointing MPI programs.

Conventionally, periodic checkpointing according to Young-Daly formula: $\tau = \sqrt{2 \cdot \delta \cdot \text{MTTF}}$. This assumes that failure arrival process is exponentially distributed.

However, this may not always hold true. Clearly, from Figure 1, the CDF is not exponential.

Much more strongly "bath-tub". Maybe can use exponentiated Weibull to model the failures.

Regardless, because it is not memoryless, the checkpointing interval cannot be uniform. Maybe can use [5], which uses dynamic programming and also comes with a simulator of sorts.

6 SCISPOT INTERFACE AND IMPLEMENTATION

Central controller.

Cloud APIs for launching the jobs.

Slurm.

Job groups are specified via a JSON file that we then use to generate different parameter combinations.

Example of a JSON file here? What about the description? Maybe some details about ranges and fixed values?

7 EVALUATION

The contenders:

(1) Run every job on un-tuned on-demand instance (cost and running time)
(2) Run every job on nanohub/big-red-2 (running and waiting time)
(3) Run on transient, restart every time (cost)
(4) SciSpot with early stopping and job sacrificing

Performance of 3 benchmarks on different types of instance types and bigred2.

7.1 Preemption likelihood curves

7.2 Searching for the best cloud configuration

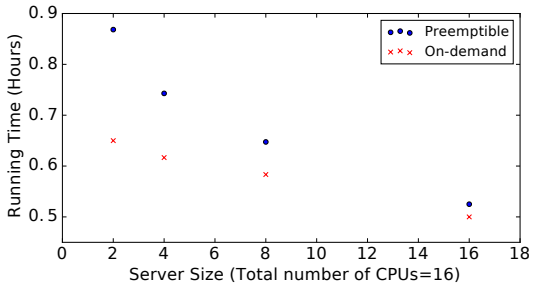

Figure 2: confinement running times

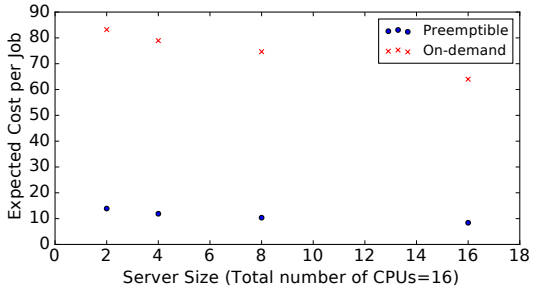

Figure 3: confinement cost

7.3 Total cost vs. running time graphs

7.4 HPC

8 RELATED WORK

8.1 Scientific applications on cloud

A classic survey is [14] [38]

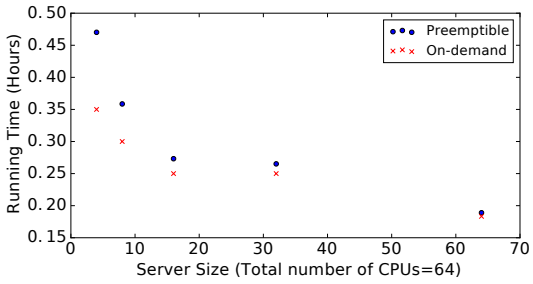

Figure 4: confinement running times

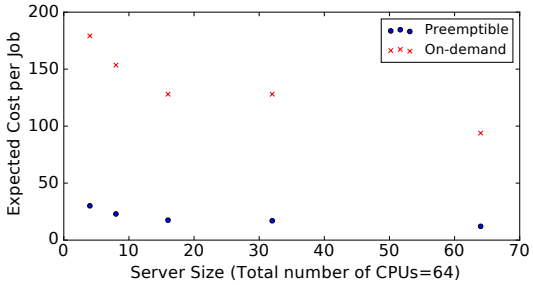

Figure 5: confinement cost

Figure 6: Ratio of waiting time to job running time on an HPC cluster. Average is 0.2

Parameter sweep: [6]

Price optimizations for Scientific workflows in the cloud [11]

8.2 Transiency mitigation

[27] classic work on MPI and Spot. Uses checkpointing. Redundancy, but for what? User specified number of VMs. Does not do instance selection. BCLR for checkpointing.

MOre spot and MPI: [12]. FOcussed on bidding and checkpoint interval. But bidding doesnt matter.

[33] is early work for spot and MPI and

[32] a batch computing service

Heterogenity often used, but not useful in the context of MPI jobs [29]

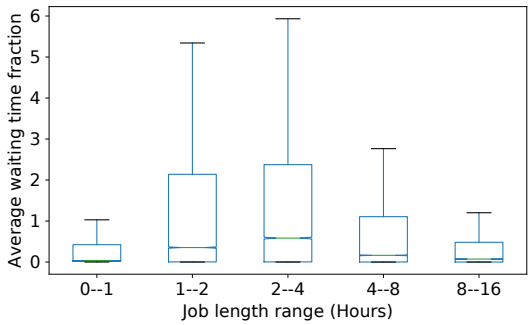

Figure 7: Waiting time fraction of jobs of different lengths varies.

Selecting the best instance type, often for data analysis computations [2], and [37], and others like Ernest and Hemingway.

All the past work was on EC2 spot market with gang failures and independent markets [12, 27]. However this assumption has now changed, and failures can happen anytime. Our failure model is more general, and applies to both cases.

8.2.1 Fault-tolerance for MPI. [8] has a discussion of checkpointing frequency which is comprehensive.

Replication is another way [34]

8.2.2 Huge amount of work on bidding in HPC. [36] [13]

8.3 Server Selection

Exploring a large configuration space using bayesian optimization methods in CherryPick [**?**] and Metis [**?**].

Can also use Latin Hypercube sampling for parameter exploration?

REFERENCES

[1] Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Tech. Rep. LLNL-TR-490254.

[2] Alipourfard, O., and Yu, M. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. 15.

[3] Bartók, A. P., De, S., Poelking, C., Bernstein, N., Kermode, J. R., Csányi, G., and Ceriotti, M. Machine learning unifies the modeling of materials and molecules. *Science Advances 3*, 12 (2017).

[4] Botu, V., and Ramprasad, R. Adaptive machine learning framework to accelerate ab initio molecular dynamics. *International Journal of Quantum Chemistry 115*, 16 (2015), 1074–1083.

[5] Bougeret, M., Casanova, H., Rabie, M., Robert, Y., and Vivien, F. Checkpointing strategies for parallel jobs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11* (Seattle, Washington, 2011), ACM Press, p. 1.

[6] Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)* (May 2000), pp. 349–363.

[7] Ch'ng, K., Carrasquilla, J., Melko, R. G., and Khatami, E. Machine learning phases of strongly correlated fermions. *Phys. Rev. X 7* (Aug 2017), 031038.

[8] Dongarra, J., Herault, T., and Robert, Y. Fault tolerance techniques for high-performance computing. 66.

[9] Ferguson, A. L. Machine learning and data science in soft materials engineering. *Journal of Physics: Condensed Matter 30*, 4 (2017), 043002.

[10] Fox, G., Glazier, J. A., Kadupitiya, J., Jadhao, V., Kim, M., Qiu, J., Sluka, J. P., Somogyi, E., Marathe, M., Adiga, A., et al. Learning everywhere: Pervasive machine learning for effective high-performance computation. *arXiv preprint arXiv:1902.10810* (2019).

[11] Garí, Y., Monge, D. A., Mateos, C., and García Garino, C. Learning budget assignment policies for autoscaling scientific workflows in the cloud. *Cluster Computing* (Feb. 2019).

[12] Gong, Y., He, B., and Zhou, A. C. Monetary cost optimizations for MPI-based HPC applications on Amazon clouds: checkpoints and replicated execution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15* (Austin, Texas, 2015), ACM Press, pp. 1–12.

[13] Guo, W., Chen, K., Wu, Y., and Zheng, W. Bidding for Highly Available Services with Low Price in Spot Instance Market. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '15* (Portland, Oregon, USA, 2015), ACM Press, pp. 191–202.

[14] Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. H. J. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems 22*, 6 (June 2011), 931–945.

[15] Jadhao, V., Solis, F. J., and Olvera de la Cruz, M. Simulation of charged systems in heterogeneous dielectric media via a true energy functional. *Phys. Rev. Lett. 109* (Nov 2012), 223905.

[16] Jadhao, V., Solis, F. J., and Olvera de la Cruz, M. A variational formulation of electrostatics in a medium with spatially varying dielectric permittivity. *The Journal of Chemical Physics 138*, 5 (2013), 054119.

[17] Jadhao, V., Thomas, C. K., and Olvera de la Cruz, M. Electrostatics-driven shape transitions in soft shells. *Proceedings of the National Academy of Sciences 111*, 35 (2014), 12673–12678.

[18] Jadhao, V., Yao, Z., Thomas, C. K., and De La Cruz, M. O. Coulomb energy of uniformly charged spheroidal shell systems. *Physical Review E 91*, 3 (2015), 032305.

[19] Jing, Y., Jadhao, V., Zwanikken, J. W., and Olvera de la Cruz, M. Ionic structure in liquids confined by dielectric interfaces. *The Journal of chemical physics 143*, 19 (2015), 194508.

[20] Kadupitiya, J., Fox, G., and Jadhao, V. *Submitted* (2018).

[21] Kadupitiya, J., Fox, G., and Jadhao, V. Machine learning for performance enhancement of molecular dynamics simulations. Accepted.

[22] Kadupitiya, J., Marru, S., Fox, G. C., and Jadhao, V. Ions in nanoconfinement, Dec 2017. Online on nanoHUB; source code on GitHub at github.com/softmaterialslab/nanoconfinement-md.

[23] Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B. L., Cohen, J., DeVito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., and Still, C. Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)* (Boston, USA, May 2013).

[24] Karlin, I., Keasler, J., and Neely, R. Lulesh 2.0 updates and changes. Tech. Rep. LLNL-TR-641973, August 2013.

[25] Liu, J., Qi, Y., Meng, Z. Y., and Fu, L. Self-learning monte carlo method. *Phys. Rev. B 95* (Jan 2017), 041101.

[26] Long, A. W., Zhang, J., Granick, S., and Ferguson, A. L. Machine learning assembly landscapes from particle tracking data. *Soft Matter 11*, 41 (2015), 8141–8153.

[27] Marathe, A., Harris, R., Lowenthal, D., De Supinski, B. R., Rountree, B., and Schulz, M. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *HPDC* (2014), ACM.

[28] Schoenholz, S. S. Combining machine learning and physics to understand glassy systems. *Journal of Physics: Conference Series 1036*, 1 (2018), 012021.

[29] Sharma, P., Irwin, D., and Shenoy, P. Portfolio-driven resource management for transient cloud servers. In *Proceedings of ACM Measurement and Analysis of Computer Systems* (June 2017), vol. 1, p. 23.

[30] Solis, F. J., Jadhao, V., and De La Cruz, M. O. Generating true minima in constrained variational formulations via modified lagrange multipliers. *Physical Review E 88*, 5 (2013), 053306.

[31] Spellings, M., and Glotzer, S. C. Machine learning for crystal identification and discovery. *AIChE Journal 64*, 6 (2018), 2198–2206.

[32] Subramanya, S., Guo, T., Sharma, P., Irwin, D., and Shenoy, P. SpotOn: A Batch Computing Service for the Spot Market. In *SOCC* (August 2015).

[33] Taifi, M., Shi, J. Y., and Khreishah, A. SpotMPI: A Framework for Auction-Based HPC Computing Using Amazon Spot Instances. In *Algorithms and Architectures for Parallel Processing*, Y. Xiang, A. Cuzzocrea, M. Hobbs, and W. Zhou, Eds., vol. 7017. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 109–120.

[34] Walters, J. P., and Chaudhary, V. Replication-Based Fault Tolerance for MPI Applications. *IEEE Transactions on Parallel and Distributed Systems 20*, 7 (July 2009), 997–1010.

[35] Ward, L., Dunn, A., Faghaninia, A., Zimmermann, N. E., Bajaj, S., Wang, Q., Montoya, J., Chen, J., Bystrom, K., Dylla, M., et al. Matminer: An open source toolkit for materials data mining. *Computational Materials Science 152* (2018), 60–69.

[36] Wolski, R., Brevik, J., Chard, R., and Chard, K. Probabilistic guarantees of execution duration for Amazon spot instances. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17* (Denver, Colorado, 2017), ACM Press, pp. 1–11.

[37] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. Selecting the *best* VM across multiple public clouds: a data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17* (Santa Clara, California, 2017), ACM Press, pp. 452–465.

[38] Zhai, Y., Liu, M., Zhai, J., Ma, X., and Chen, W. Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications. In *State of the Practice Reports on - SC '11* (Seattle, Washington, 2011), ACM Press, p. 1.